IFPark - Um sistema de gerenciamento de vagas para os estacionamentos do IFBA

Trabalho de Conclusão de Curso

Luis André Sousa Sousa

Manoel Carvalho Marques Neto
Orientador

Instituto Federal da Bahia - IFBA
Curso de Análise e Desenvolvimento de Sistemas
Campus Salvador

Salvador, Bahia, Brasil
21 de janeiro de 2026

Sumário

1 Visão geral

1.1 Declaração do Problema

A gestão de estacionamentos em instituições de ensino superior de grande porte, representa um grande desafio logístico e operacional que impacta diretamente a rotina da comunidade acadêmica. O Instituto Federal da Bahia, mais precisamente no Campus Salvador, exemplifica esta realidade devido a combinação de alguns fatores.

O campus está situado no bairro do Barbalho, uma região de logística um tanto complexa[1], com vias de mão única e um ponto de ônibus estratégico em frente à instituição, que frequentemente causa engarrafamentos e tumultos na região. Além disso, o fluxo constante de veículos da comunidade acadêmica (estudantes, professores, funcionários e visitantes) gera uma alta demanda por um número limitado de vagas no estacionamento.

A paralisação do fluxo ocorre quando os motoristas encontram os estacionamentos lotados. Eles permanecem parados, esperando por uma vaga disponível, e isso, consequentemente, causa a formação de filas na entrada dos estacionamentos. Este comportamento ocorre porque não há, atualmente, nenhum sistema que informe a disponibilidade de vagas. O motorista só descobre a lotação após já ter chegado ao campus, e sua decisão de esperar gera uma agitação frequente.

Esse cenário problemático atual é agravado pelo controle de acesso que é inteiramente manual. Basicamente, um vigia ou porteiro registra as entradas e saídas dos veículos em um caderno de anotações. Este método possui diversas desvantagens:

- Congestionamento: A verificação manual de credenciais ou o preenchimento de formulários em papel na entrada do campus pode gerar lentidão e filas, especialmente em horários de pico.
- Falta de informação: A informação em um caderno não é centralizada nem facilmente acessível. Os motoristas não têm visibilidade sobre a disponibilidade de vagas, levando a um tempo excessivo gasto esperando por uma vaga ou circulando pela região ao redor do campus em busca de um local, o que aumenta o estresse[2] e o consumo de combustível[3] e o consumo de combustível.
- Vulnerabilidades de segurança: Um controle baseado em papel é suscetível a falhas, rasuras, perdas e pode dificultar a distinção entre membros autorizados da comunidade acadêmica e visitantes não registrados, além de não manter um histórico confiável de acessos.
- Ausência de dados para gestão: A falta de registros digitais, que poderiam formar uma trilha de auditoria confiável [4], impede que a administração da instituição realize análises estratégicas sobre a demanda, ocupação e necessidade de expansão ou readequação da infraestrutura.

A motivação para o desenvolvimento deste trabalho nasce da observação direta e diária

dessas dificuldades no nosso ambiente acadêmico. Notoriamente, existe a necessidade de modernizar essa infraestrutura. A implementação de um sistema digital não apenas soluciona as ineficiências operacionais, mas também fornece uma ferramenta de gestão estratégica, melhora a segurança e eleva a qualidade da experiência diária de toda a comunidade acadêmica no campus.

1.2 Proposta de Solução de Software

Este trabalho propõe o desenvolvimento de um sistema digital focado na gestão ágil dos estacionamentos. A solução será composta por dois componentes principais:

1. API RESTful (backend): Esta API centralizará todas as regras de negócio, a segurança (autenticação e autorização) e a interação com o banco de dados.

2. Aplicativo *mobile* (frontend): Esta será a ferramenta de trabalho do vigia. A aplicação vai consumir a API e permitir que o vigia substitua o caderno por um dispositivo móvel.

O sistema vai constar com as seguintes funcionalidades:

- Controle de acesso ágil: O porteiro utilizará o aplicativo mobile para consultar um motorista (pela placa, nome ou matrícula) e registrar sua entrada com poucos toques, liberando o acesso.

- Cadastro prévio e identificação: O sistema vai manter um cadastro de motoristas, funcionários e veículos autorizados. Atualmente, o cadastro é feito presencialmente no prédio administrativo da instituição. Com o aplicativo, tanto o cadastro quanto a aprovação daquele usuário poderão ser feitos de forma *online*, tornando o processo mais dinâmico. O aplicativo também permitirá o registro controlado de ”visitantes”, operado pelo porteiro.

- Gerenciamento de vagas: Ao registrar a entrada, o sistema decrementa o contador de vagas daquele estacionamento específico. Na saída, o processo inverso atualiza o contador. A contagem exata de vagas estará visível no sistema em tempo integral para os motoristas, porteiros e administradores. Inclusive esse dado poderá ser transmitido para um letreiro digital ou monitor, caso a instituição opte por esse recurso.

- Estrutura multicampi: A modelagem de dados (com entidades de Campus e Estacionamento) e um tipo de usuário especial (super-administrador) permitirá que a solução seja escalável para outros campi do IFBA.

Esse sistema substitui o processo atual por um fluxo digital moderno e dinâmico via aplicativo. Isso contribui para minimizar falhas de segurança e para uma melhor gestão dos dados. Além disso, ataca diretamente o problema das filas, pois os motoristas saberão exatamente quantas vagas livres existem. Isso auxilia na tomada de decisão de usar ou não o estacionamento do instituto naquele momento. No geral, o software pode melhorar a experiência de toda a comunidade acadêmica que está atrelada aos estacionamentos do campus.

1.3 Tecnologias adotadas

A escolha das tecnologias foi pautada na criação de uma arquitetura cliente-servidor moderna, separando claramente as responsabilidades do backend e do frontend.

1.3.1 Backend (API)

1. Java 17+: É uma linguagem madura, criada há quase 30 anos. Possui uma comunidade muito ativa e um ecossistema robusto para desenvolvimento web. Além desses fatores, a versão 17 garante acesso a recursos modernos da linguagem junto a estabilidade e suporte de longo prazo (LTS). [5]

2. Spring Boot 3: É o framework principal para o desenvolvimento de APIs REST em Java. Ele permite configurar a aplicação de forma rápida e simples, o que agrega positivamente na experiência de desenvolvimento. Ademais, o Spring possui uma gama muito ampla de bibliotecas compatíveis, fornecendo um ecossistema confiável e produtivo. [6]

3. Spring Security: Ferramenta padrão do ecossistema Spring para proteger o sistema de acessos indevidos. Basicamente, ela gerencia a autenticação (login) e autorização (permissões baseadas nos perfis, como ”porteiro”ou ”administrador”) utilizando tokens JWT para comunicação segura com o app mobile. [7]

4. Spring Data JPA: É uma ferramenta que abstrai e simplifica a camada de acesso ao banco de dados, permitindo a interação com o banco através de objetos Java. É também o ORM (*Object Relational Mapping*) oficial do Spring. [8]

5. PostgreSQL: É um Sistema Gerenciador de Banco de Dados (SGBD) de código aberto, poderoso, confiável e amplamente usado no mercado. Ideal para suportar os dados relacionais do sistema. [9]

6. Flyway: É uma ferramenta para versionar o banco de dados. Ela permite um melhor controle sobre a estrutura do esquema do banco, facilitando inserções de tabelas novas ou modificações em tabelas já existentes. [10]

7. Springdoc-openapi (Swagger): Gera uma documentação interativa da API, permitindo visualizar e testar as requisições pelo navegador. Essa possibilidade torna a aplicação mais acessível aos usuários e desenvolvedores. [11]

1.3.2 Frontend (aplicativo para dispositivos móveis)

1. React Native: Sua principal vantagem é a capacidade de criar aplicações nativas para os sistemas operacionais Android e iOS a partir de um único código-base escrito em Javascript ou Typescript. Além disso, sua comunidade e disponibilidade de bibliotecas facilitam e otimizam o tempo de desenvolvimento. [12]

2. React Native ML Kit: Esta biblioteca, que integra o kit de *Machine Learning* do Google, permite que o aplicativo utilize a câmera do dispositivo para escanear e automaticamente extrair o texto. A ideia é usar essa ferramenta para implementar a funcionalidade de

identificar os motoristas por meio de imagens da placa. [13]

3. React Navigation: É a solução padrão da comunidade para gerenciar a transição entre diferentes telas. Permite a navegação usando mecanismos de pilhas e abas, semelhante aos navegadores da web. Ela também oferece gestos e animações padrões do Android e do iOS ao navegar pelo aplicativo. [14]

4. React Native Async Storage: Utilizada para persistir dados não-sensíveis no dispositivo, como preferências do usuário, cache de informações (ex: último e-mail utilizado no login) ou dados de configuração. Funciona como um armazenamento simples de chave-valor. [15]

5. React Native Keychain: Esta biblioteca fornece uma interface para o armazenamento seguro de credenciais (como tokens) utilizando os mecanismos nativos e criptografados de cada plataforma — o Keychain no iOS e o Keystore no Android. O uso desta biblioteca é uma prática de segurança essencial para evitar o armazenamento de dados sensíveis em locais de fácil acesso, como o AsyncStorage. [16]

6. React Hook Form e Zod: São bibliotecas que ajudam no gerenciamento de dados dos formulários e validação de esquemas respectivamente. A integração entre as duas permite definir regras claras no frontend (exemplo: ”o campo PLACA é obrigatório e deve ter 7 caracteres”). Isso garante que apenas dados válidos sejam enviados para a API, reduzindo a carga no servidor e melhorando o *feedback* ao usuário. [17, 18]

1.3.3 Ferramentas Auxiliares

1. Git: Ferramenta padrão para o gerenciamento do código-fonte, rastreamento de alterações e manutenção do histórico. [19]

2. IntelliJ IDEA: É uma IDE popular para o desenvolvimento com Java. Ela consegue monitorar tanto a sintaxe da linguagem como também extensões e outras ferramentas, incluindo funcionalidades do Spring e arquivos de configurações do projeto. [20]

3. Visual Studio Code: É um editor de código que se tornou uma ferramenta popular, principalmente no ecossistema JavaScript. Possui acesso a um número vasto de extensões que aumentam a produtividade do desenvolvimento. [21]

4. Bruno: É um cliente de API de código aberto, utilizado para testar e depurar os endpoints da aplicação. Diferencia-se de outras ferramentas por armazenar as coleções de requisições diretamente no sistema de arquivos, facilitando o versionamento junto ao código do projeto via Git. [22]

2 Requisitos

2.1 Requisitos Funcionais

RF1 Como um motorista, eu posso realizar um auto-cadastro com meus dados pessoais.

RF2 Como um motorista, eu posso acrescentar os dados do meu veículo no meu cadastro.

RF3 Como um usuário, eu posso ser informado pelo aplicativo quando meu cadastro for aprovado ou rejeitado por um administrador.

RF4 Como um usuário, eu posso fazer meu login no aplicativo móvel usando minhas credenciais.

RF5 Como um usuário, eu posso ver no aplicativo a contagem de vagas livres e ocupadas para cada estacionamento do meu campus.

RF6 Como um porteiro, eu posso consultar a lista de todos os motoristas e veículos cadastrados.

RF7 Como um porteiro, eu posso registrar a entrada de um veículo cadastrado (buscando pela placa).

RF8 Como um porteiro, eu posso usar a câmera do celular para escanear a placa de um veículo e agilizar o registro de entrada.

RF9 Como um porteiro, eu posso ser impedido ou alertado pelo sistema ao tentar registrar uma entrada se o estacionamento estiver lotado.

RF10 Como um porteiro, eu posso registrar a entrada de um visitante não-cadastrado informando a placa do veículo.

RF11 Como um porteiro, eu posso visualizar uma lista de todas movimentações de veículos em um estacionamento do meu campus.

RF12 Como um porteiro, eu posso registrar a saída de um veículo (buscando pela placa, nome ou matrícula).

RF13 Como um administrador, eu posso cadastrar, editar e desativar usuários associados ao meu campus.

RF14 Como um administrador, eu posso cadastrar, editar e desativar os estacionamentos pertencentes ao meu campus.

RF15 Como um administrador, eu posso aprovar ou rejeitar novos cadastros de motoristas.

RF16 Como um admminstrador, eu posso cadastrar, editar e desativar motoristas e seus veículos manualmente.

RF17 Como um super-administrador, eu posso cadastrar, editar e desativar campi no sistema.

RF18 Como um super-administrador, eu posso cadastrar, editar e desativar usuários do tipo admminstrador.

RF19 Como um super-administrador, eu posso associar um usuário administrador a um

campus específico.

2.2 Requisitos Não-Funcionais

RNF1 O sistema deve ser acessível pelos vigias e administradores através de uma aplicação *mobile*.

RNF2 A API backend deve ter um tempo de resposta máximo de 2 segundos para operações de registro de entrada ou saída de veículos e de consulta de vagas.

RNF3 O sistema deve garantir a autenticação segura de usuários via login e senha com token JWT e autorização baseada nos papéis (motorista, vigia, administrador, super-administrador).

RNF4 A interface da aplicação móvel deve ser intuitiva, permitindo que um vigia com treinamento mínimo possa operar o sistema de forma eficiente.

RNF5 A API backend deve ser desenvolvida em Java com Spring Boot, e a aplicação móvel em Typescript com React Native.

RNF6 A API deve possuir um template específico de retorno para os casos de erro e também para os casos em que são necessários informar algo sobre a requisição.

RNF7 O sistema deve ser protegido contra acessos não autorizados.

RNF8 O sistema deve realizar conexão com o banco de dados relacional PostgresSQL.

RNF9 Todos os dados sensíveis de usuários (como senhas e tokens) devem ser armazenados de forma criptografada.

RNF10 O sistema deve funcionar corretamente em conexões de dados móveis (4G/5G) com latência moderada, comum em ambientes de campus.

3 Design

3.1 Projeto UML

3.1.1 Diagrama de Classe

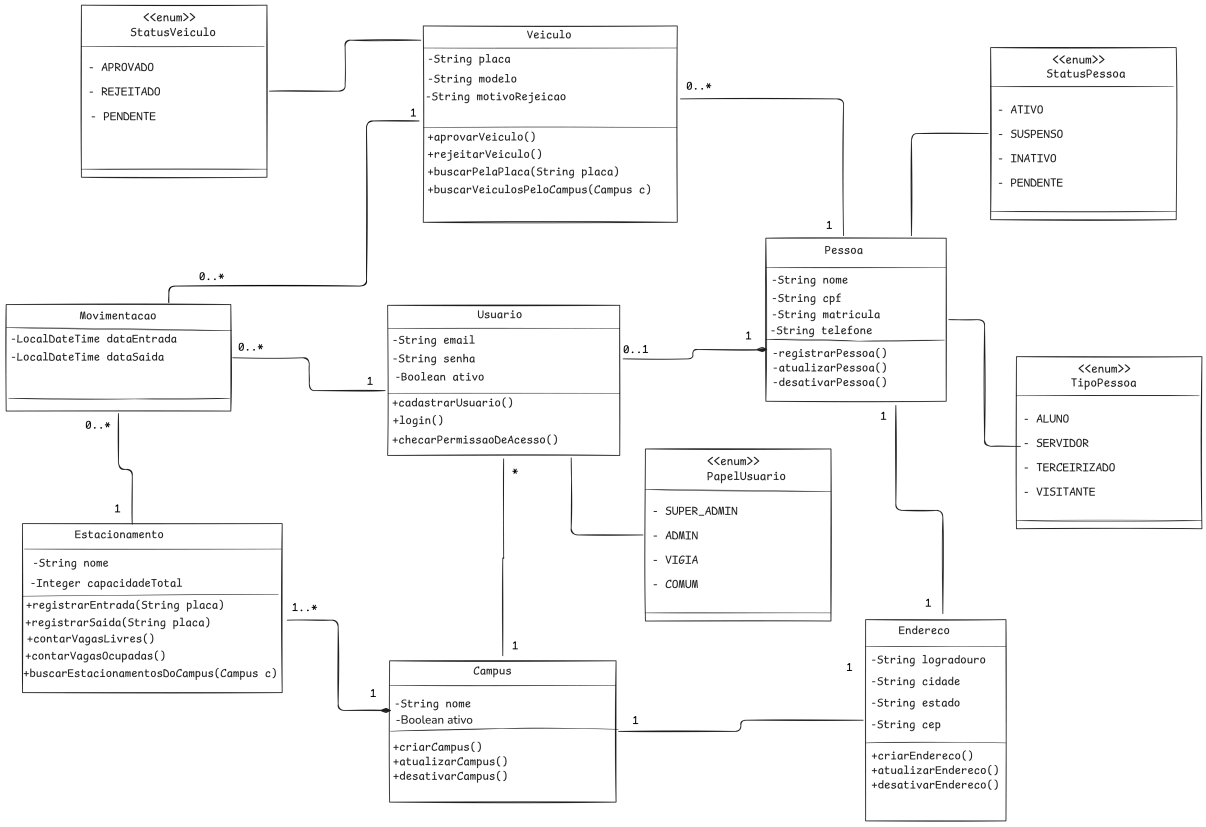

Figura 1: Diagrama de Classe do Projeto

3.1.2 Diagramas de Caso de Uso

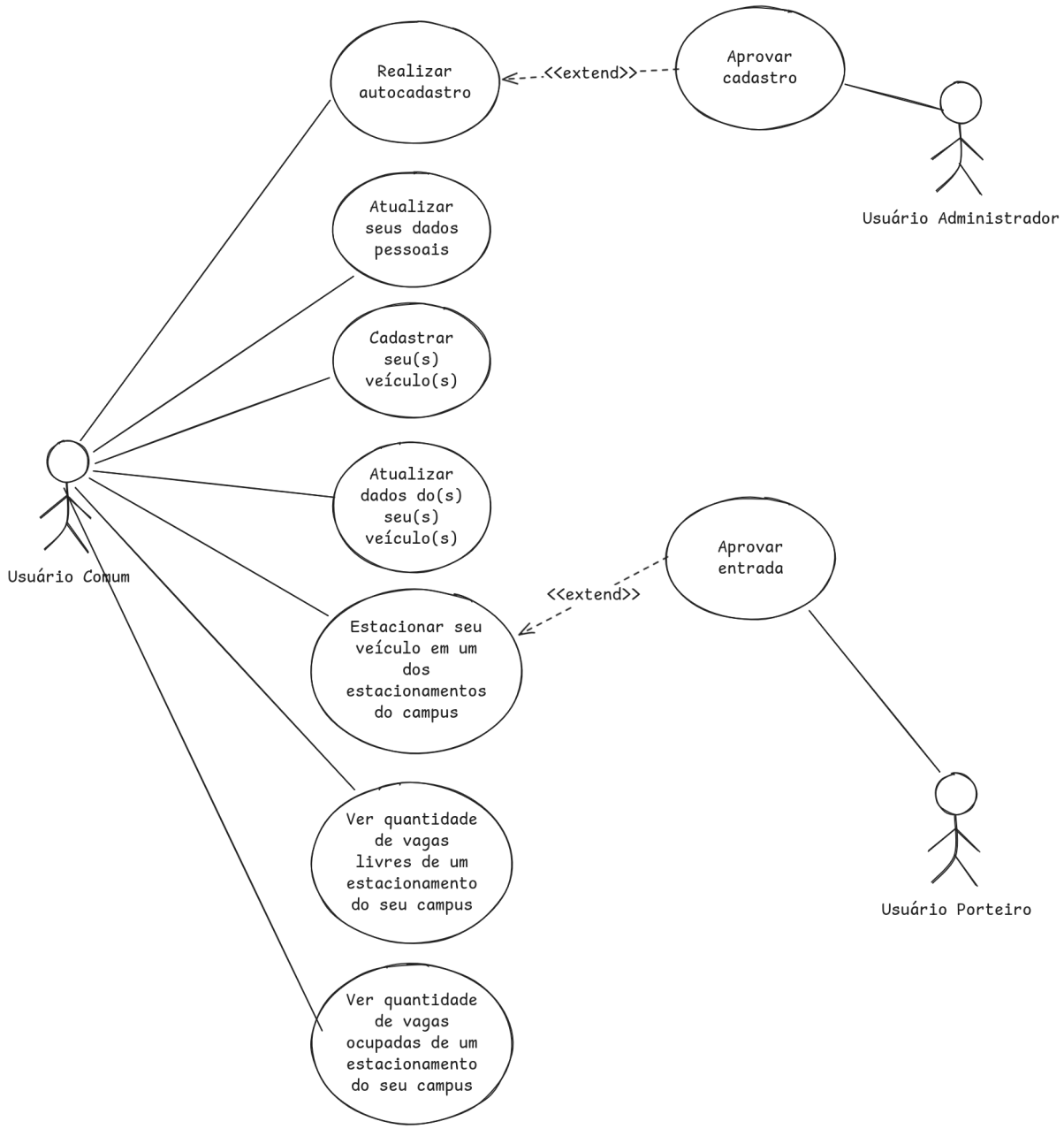

Figura 2: Diagrama de Caso de Uso do Usuário Comum

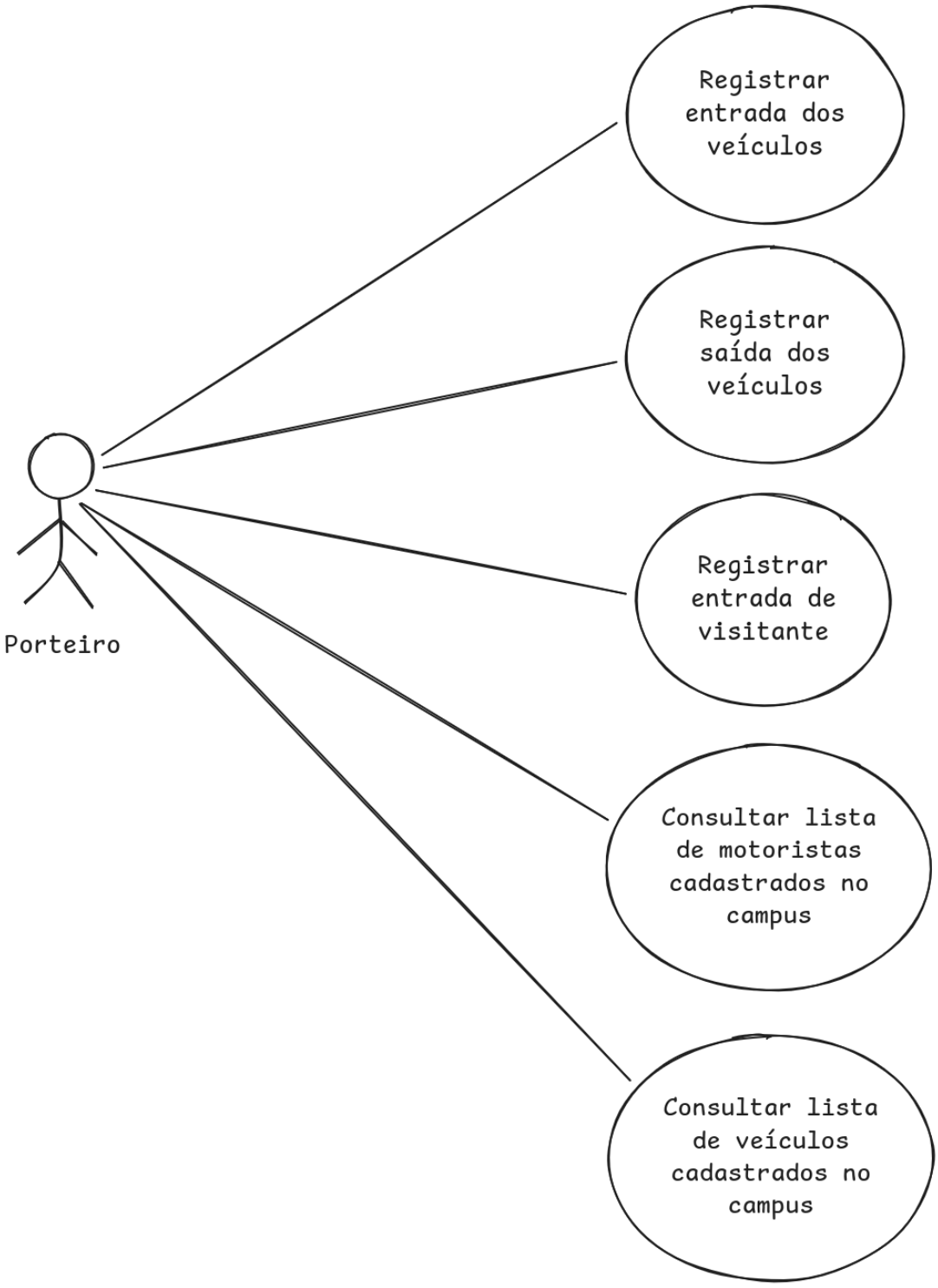

Figura 3: Diagrama de Caso de Uso do Porteiro

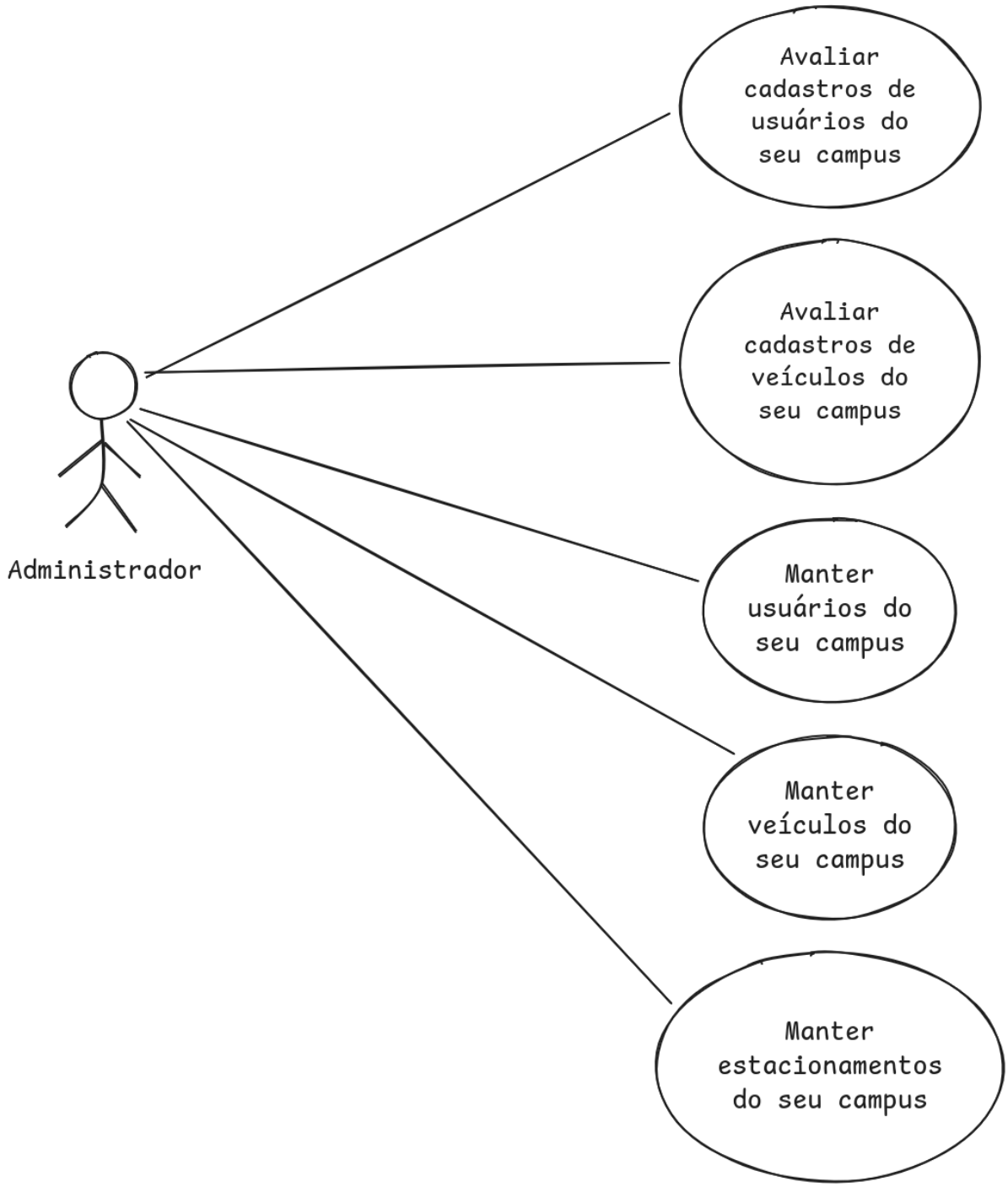

Figura 4: Diagrama de Caso de Uso do Administrador

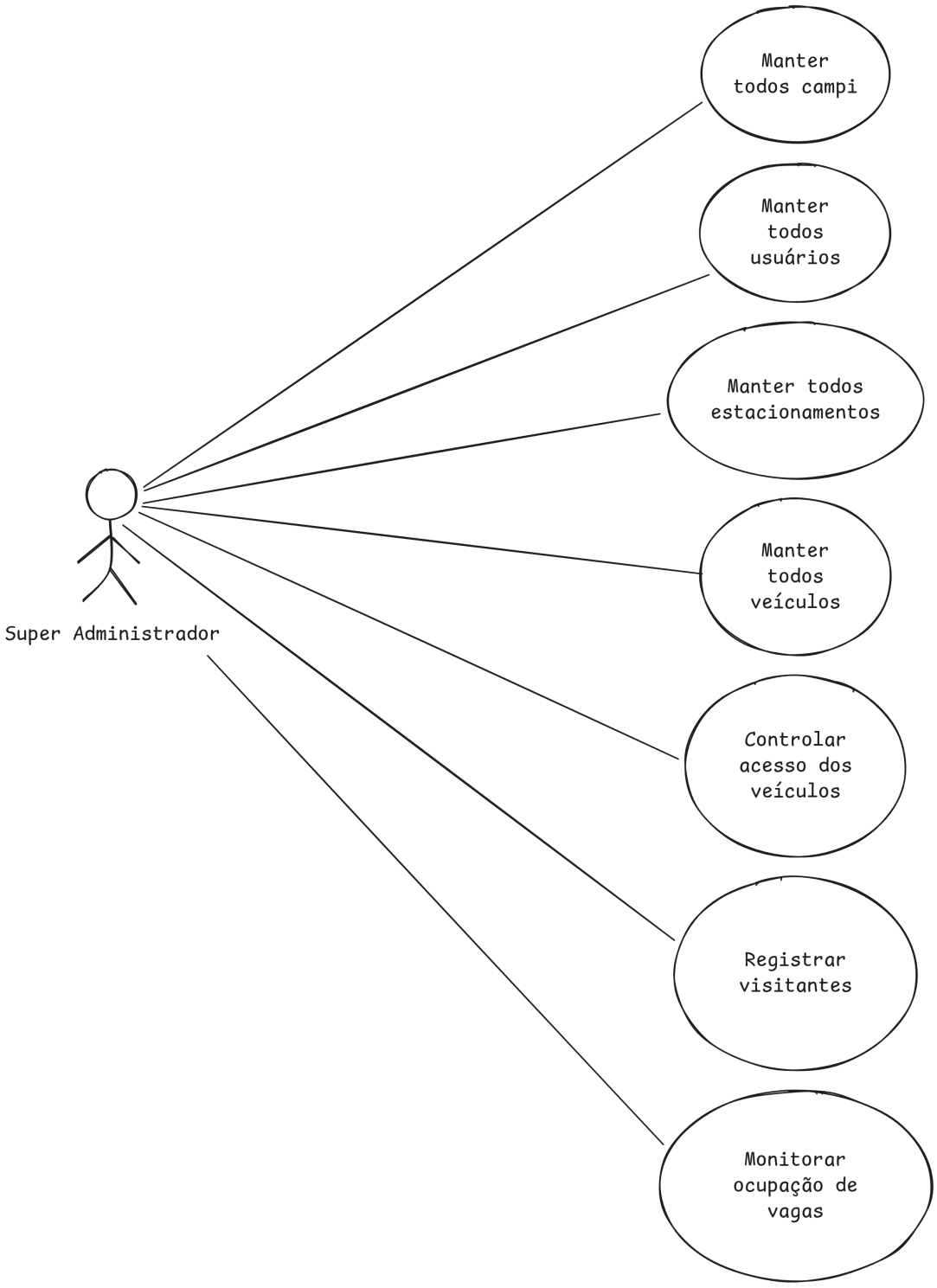

Figura 5: Diagrama de Caso de Uso do Super Administrador

3.2 Visão arquitetural

Como mencionado antes, a arquitetura do sistema baseia-se no modelo cliente-servidor, adotando o estilo arquitetural REST. Isso garante a separação de responsabilidades entre as camadas de apresentação e de processamento. O ecossistema é composto por uma aplicação móvel atuando como frontend e uma API servindo como backend, comunicando-se de forma assíncrona.

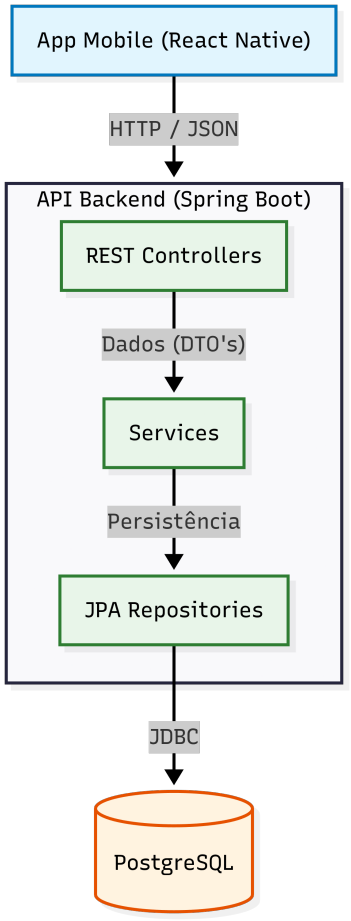

Figura 6: Visão Arquitetural do Sistema

Na camada de apresentação, utiliza-se o framework React Native para o desenvolvimento da aplicação móvel. Esta camada é responsável exclusivamente pela interação com o usuário final e pela visualização dos dados. A comunicação com o servidor ocorre através do protocolo HTTP, utilizando o formato JSON (JavaScript Object Notation) para a serialização e transporte de dados.

O backend foi desenvolvido sobre o ecossistema Spring Boot, estruturado em uma arquitetura em camadas (Layered Architecture). As requisições são interceptadas inicialmente pelos *Controllers*, que atuam como pontos de entrada da API, gerenciando os verbos HTTP e direcionando essas solicitações para a lógica adequada. Para garantir a segurança e a integridade da estrutura de dados interna, utiliza-se o padrão DTO (Data Transfer Object), que isola as entidades de domínio das interfaces externas, filtrando os dados que trafegam entre

o cliente e o servidor.

A lógica de negócios reside na camada de *Services*. É neste nível que são processadas as regras cruciais do sistema de estacionamento, como validação de acesso, cálculo de permanência e gerenciamento de vagas. Após o processamento, a persistência dos dados é gerenciada pela camada de *Repositories*. Utilizando a especificação Jakarta Persistence API (JPA) e a abstração do Spring Data, o sistema realiza operações de banco de dados de forma orientada a objetos, eliminando a necessidade de SQL verboso para operações padrão.

Por fim, a camada de persistência conecta-se via JDBC (Java Database Connectivity) ao Sistema Gerenciador de Banco de Dados Relacional PostgreSQL. É aqui onde são armazenadas as informações persistentes de veículos, usuários e os registros de movimentação.

3.3 Modelo de Banco de Dados

3.3.1 Modelo de Dados Lógico

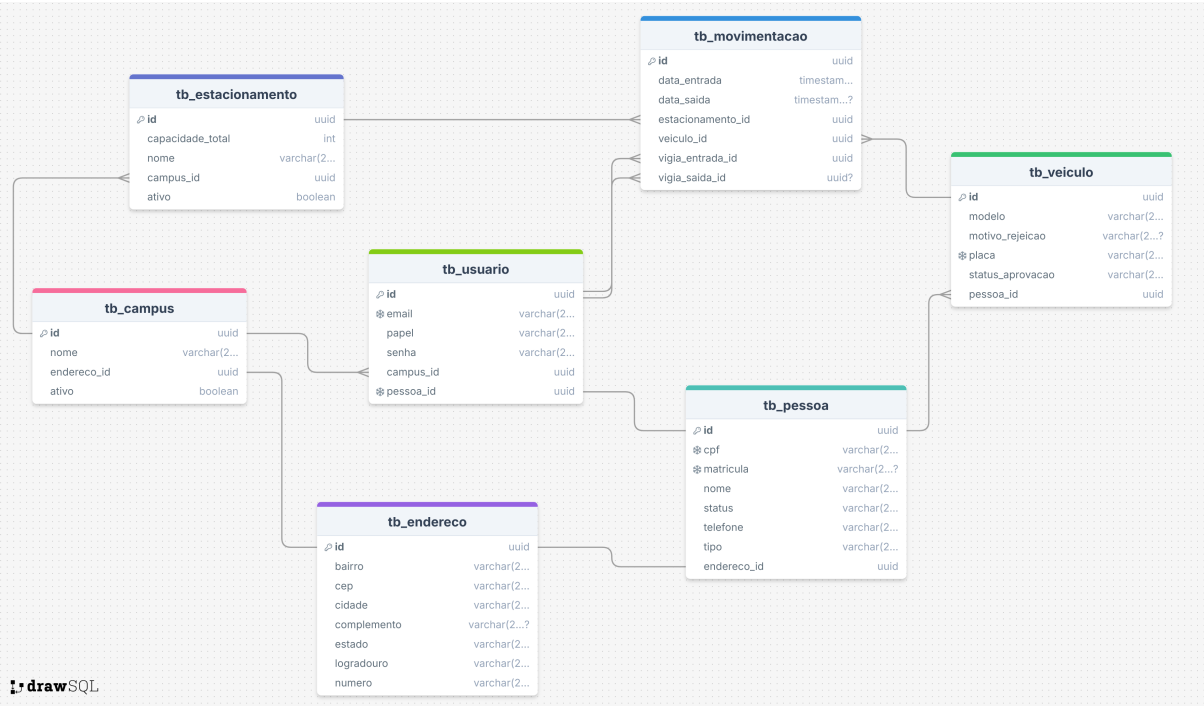

Figura 7: Diagrama do Banco de Dados

O Modelo Lógico de Dados exibe uma visão abstrata de como os dados se organizam e se relacionam no sistema. Sua implementação é independente de banco de dados, fornecendo uma visão futura para a criação do modelo físico. O diagrama da Figura 7 apresenta uma estrutura normalizada, visando reduzir redundâncias e facilitar a manutenção dos dados.

A entidade central do modelo é a `Movimentação`. Ela atua como uma tabela associativa, registrando o fluxo de entrada e saída. Ela conecta o `Veículo` que está estacionando, o

Estacionamento onde o evento ocorre e os Usuários (vigias) responsáveis por registrar a entrada e a saída.

Também houve uma decisão de separar os dados de login (Usuários) dos dados pessoais (Pessoa). Isso permite que uma pessoa física exista no banco de dados com seus veículos e documentos (CPF, Matrícula), independentemente de possuir credenciais de acesso ao sistema (login/senha/papel). Isso é importante para contemplar alguns casos especiais, como visitantes ou pessoas idosas que usam o estacionamento, mas preferem não utilizarem o sistema.

Já a entidade Endereço foi isolada para servir tanto a Pessoa quanto a Campus. Isso evita a repetição de colunas de endereço em múltiplas tabelas, aderindo às formas normais de banco de dados.

O modelo também busca refletir a estrutura física de uma instituição, onde um Campus pode possuir múltiplos Estacionamento (relação 1:N).

3.3.2 Modelo de Dados Físico

O modelo físico foi projetado para implementação no SGBD PostgreSQL. A convenção de nomenclatura adotada utiliza o padrão *snake_case* com o prefixo tb_.

1. tb_pessoa: Armazena os dados biográficos dos indivíduos vinculados à instituição.
    - id (UUID): Chave primária. Identificador único do registro.
    - nome (Varchar): Nome completo da pessoa.
    - cpf (Varchar): Documento de identificação nacional (Cadastro de Pessoas Físicas).
    - matricula (Varchar): Identificador institucional para docentes e discentes. É um campo opcional.
    - telefone (Varchar): Número de contato telefônico ou celular.
    - tipo (Varchar): Categoriza a pessoa (ex: ”Aluno”, ”Servidor”, ”Terceirizado”), fundamental para regras de prioridade.
    - status (Varchar): Indica a situação cadastral (ex: ”Ativo”, ”Inativo”).
    - endereco_id (UUID): Chave estrangeira que vincula a pessoa ao seu endereço residencial.
2. tb_usuario: Gerencia as credenciais de quem acessa o sistema digitalmente (App ou Web).
    - id (UUID): Chave primária. Identificador único do usuário.
    - email (Varchar): Endereço de e-mail utilizado como *login* de acesso.
    - senha (Varchar): Armazena o *hash* criptografado da senha.

- `papel` (Varchar): Define o nível de permissão no sistema (ex: ”ROLE_ADMIN”, ”ROLE_VIGIA”).
- `campus_id` (UUID): Chave estrangeira indicando a qual campus este usuário (porteiro/admin) está vinculado.
- `pessoa_id` (UUID): Chave estrangeira que conecta o login aos dados da pessoa física.

3. `tb_veiculo`: Armazena os dados dos automóveis cadastrados.
   - `id` (UUID): Chave primária. Identificador único do veículo.
   - `placa` (Varchar): Identificador visual do veículo, utilizado nas buscas operacionais.
   - `modelo` (Varchar): Descrição do veículo (marca/modelo/cor) para conferência visual.
   - `status_aprovacao` (Varchar): Campo de controle de fluxo (ex: ”Pendente”, ”Aprovado”).
   - `motivo_rejeicao` (Varchar): Campo descritivo opcional, preenchido apenas se a aprovação for negada.
   - `pessoa_id` (UUID): Chave estrangeira identificando o proprietário ou condutor principal do veículo.
4. `tb_campus`: Entidade raiz que representa a unidade física do IFBA.
   - `id` (UUID): Chave primária. Identificador único do campus.
   - `nome` (Varchar): Nome da unidade (ex: ”Campus Salvador”, ”Campus Feira de Santana”).
   - `endereco_id` (UUID): Chave estrangeira indicando a localização física do campus.
   - `ativo` (boolean): Campo booleano para informar se a entidade está ativa ou não.
5. `tb_estacionamento`:
   - `id` (UUID): Chave primária. Identificador único do estacionamento.
   - `nome` (Varchar): Identificação do setor (ex: ”Estacionamento Principal”, ”Estacionamento Motos”).
   - `capacidade_total` (Int): Número inteiro que define o limite físico de vagas para controle de lotação.
   - `campus_id` (UUID): Chave estrangeira que vincula este estacionamento a um campus específico.
   - `ativo` (boolean): Campo booleano para informar se a entidade está ativa ou não.

6. `tb_endereco`: Entidade normalizada para armazenar dados de logradouro.
    - `id` (UUID): Chave primária. Identificador único do endereço.
    - `logradouro` (Varchar): Nome da rua, avenida ou via.
    - `numero` (Varchar): Número do imóvel.
    - `complemento` (Varchar): Informações adicionais. Campo opcional.
    - `bairro` (Varchar): Nome do bairro.
    - `cidade` (Varchar): Nome da cidade.
    - `estado` (Varchar): Sigla ou nome da unidade federativa (UF).
    - `cep` (Varchar): Código de Endereçamento Postal.
7. `tb_movimentacao`: Registra o fluxo de entrada e saída, conectando todas as pontas do sistema.
    - `id` (UUID): Chave primária. Identificador único da movimentação.
    - `data_entrada` (Timestamp): Data e hora exata da chegada do veículo.
    - `data_saida` (Timestamp): Data e hora da partida (pode ser nulo se o veículo ainda estiver estacionado).
    - `estacionamento_id` (UUID): Chave estrangeira indicando onde o veículo estacionou.
    - `veiculo_id` (UUID): Chave estrangeira indicando qual veículo realizou a movimentação.
    - `vigia_entrada_id` (UUID): Chave estrangeira do usuário que registrou a entrada.
    - `vigia_saida_id` (UUID): Chave estrangeira do usuário que registrou a saída.

4 Testes de Software

4.1 Projeto de Testes

A estratégia de testes adotada foi estruturada para garantir a qualidade tanto dos componentes individuais (backend e frontend) quanto da integração do sistema. Dessa forma, combinou-se testes de validação da API e testes manuais do sistema.

No primeiro momento, O objetivo foi garantir que as regras de negócio implementadas na camada de backend estivessem corretas antes da integração com a interface móvel. Sendo assim, foram feitos testes de requisição nos endpoints da API REST utilizando o programa Bruno e a interface do Swagger, validando os códigos de resposta HTTP (200, 201, 400, 401, 403) e a estrutura dos objetos JSON retornados (DTOs). A segurança dos endpoints também

foi validada, garantindo que apenas usuários autenticados e autorizados, como com perfis de ADMIN ou VIGIA, pudessem realizar determinadas operações, conforme definido nas regras de negócio e implementadas com o módulo do Spring Security.

Posteriormente, focou-se na validação das funcionalidades através do aplicativo mobile. Foram executados testes manuais em um dispositivo Android, verificando a usabilidade, a navegação entre telas e a integração com o hardware do dispositivo, especificamente no uso da câmera para o reconhecimento de placas.

4.1.1 Cenários de Testes Executados

As tabelas abaixo apresentam os principais casos de teste executados para validar os Requisitos Funcionais prioritários do sistema, cobrindo os fluxos de autenticação, controle de acesso e gestão de usuários.

Tabela 1: CT01 - Auto-cadastro e Login de Motorista

Campo	**Descrição**
Requisitos	RF1, RF2, RF4
Pré-condição	Aplicativo instalado e nenhum usuário logado.
Entradas	Dados pessoais (nome, CPF, email, senha, confirmação de senha) e dados do veículo (placa, modelo).
Procedimento	1. Acessar a opção ”Cadastre-se”no aplicativo. 2. Preencher formulário de dados pessoais e veículo. 3. Confirmar envio. 4. Tentar realizar login imediato com as credenciais criadas.
Saídas Esperadas	O sistema deve exibir mensagem de cadastro realizado com sucesso. O login deve ser impedido com a mensagem ”Aguarde aprovação”.
Resultado Obtido	Sucesso.

Tabela 2: CT02 - Notificação e Liberação de Acesso

Campo	**Descrição**
Requisitos	RF3, RF15
Pré-condição	Existir um cadastro de usuário com status ”PENDENTE”.
Entradas	Ação de aprovação por parte do Administrador.
Procedimento	1. Administrador acessa lista de usuários. 2. Aprova o cadastro do usuário do CT01. 3. Motorista tenta realizar login novamente no aplicativo.
Saídas Esperadas	O status muda para ”APROVADO”. O aplicativo deve permitir o login e direcionar para a tela inicial.
Resultado Obtido	Sucesso.

Tabela 3: CT03 - Consulta de Vagas

Campo	**Descrição**
Requisitos	RF5
Pré-condição	Usuário logado no aplicativo.
Entradas	Nenhuma (apenas navegação).
Procedimento	1. Acessar a Dashboard do aplicativo. 2. Comparar os números exibidos na tela com os dados reais do banco de dados.
Saídas Esperadas	A contagem de vagas livres e ocupadas deve ser igual à persistida no banco de dados.
Resultado Obtido	Sucesso.

Tabela 4: CT04 - Registro de Entrada com OCR

Campo	**Descrição**
Requisitos	RF6, RF7, RF8
Pré-condição	Usuário Porteiro logado e estacionamento com vagas livres.
Entradas	Imagem da placa capturada pela câmera.
Procedimento	1. Porteiro seleciona ”Registrar Entrada”. 2. Aciona o ícone de câmera e aponta para a placa do veículo. 3. Registra a imagem , aguarda o reconhecimento e confirma a entrada.
Saídas Esperadas	O OCR deve transcrever a placa corretamente. O sistema deve validar se o veículo é cadastrado, registrar a entrada e decrementar 1 vaga.
Resultado Obtido	Sucesso.

Tabela 5: CT05 - Bloqueio de Entrada por Lotação

Campo	**Descrição**
Requisitos	RF9
Pré-condição	Estacionamento com capacidade total atingida (0 vagas livres).
Entradas	Tentativa de registro de entrada de um veículo autorizado.
Procedimento	1. Porteiro seleciona o estacionamento lotado. 2. Insere a placa de um veículo válido. 3. Tenta confirmar a entrada.
Saídas Esperadas	O sistema deve bloquear a ação, exibir um alerta visual de ”Estacionamento Lotado”e não registrar a movimentação.
Resultado Obtido	Sucesso.

Tabela 6: CT06 - Entrada de Visitante Não-Cadastrado

Campo	**Descrição**
Requisitos	RF10
Pré-condição	Visitante na portaria sem cadastro prévio.
Entradas	Nome, CPF e Placa do veículo.
Procedimento	1. Porteiro seleciona ”Registrar Visitante”. 2. Preenche os dados solicitados manualmente. 3. Confirma a entrada.
Saídas Esperadas	O sistema deve criar um registro de visitante no banco, vincular a movimentação e atualizar o contador de vagas.
Resultado Obtido	Sucesso.

Tabela 7: CT07 - Registro de Saída e Histórico

Campo	**Descrição**
Requisitos	RF11, RF12
Pré-condição	Veículo presente dentro do estacionamento.
Entradas	Placa do veículo de saída.
Procedimento	1. Porteiro consulta a página de movimentações(RF11). 2. Seleciona ”Registrar Saída”. 3. Insere ou escaneia os dados da placa 4. Confirma saída
Saídas Esperadas	O registro de movimentação deve receber a data/hora de saída. O contador de vagas livres deve ser incrementado.
Resultado Obtido	Sucesso.

Tabela 8: CT08 - Gestão Administrativa (CRUD)

Campo	**Descrição**
Requisitos	RF13, RF14, RF16
Pré-condição	Usuário Administrador logado.
Entradas	Dados de atualização de um usuário ou estacionamento.
Procedimento	1. Acessar a página de um estacionamento. 2. Editar a capacidade total de um estacionamento. 3. Desativar um estacionamento. 4. Acessar página de usuários 5. Rejeitar um usuário.
Saídas Esperadas	As alterações devem ser salvas no banco. O estacionamento deve refletir a nova capacidade imediatamente no cálculo de vagas. O estacionamento desativado deve constar como ativo = false.
Resultado Obtido	Sucesso.

Tabela 9: CT09 - Configuração Multi-Campus (Super Admin)

Campo	Descrição
Requisitos	RF17, RF18, RF19
Pré-condição	Usuário Super-Administrador logado.
Entradas	Dados de novo Campus e novo Administrador Local.
Procedimento	1. Cadastrar novo Campus ”IFBA - Feira de Santana”. 2. Cadastrar novo usuário administrador. 3. Atributo ”campus”deste usuário deve se referir ao novo campus.
Saídas Esperadas	O novo administrador deve conseguir logar e visualizar apenas os dados do Campus Feira de Santana.
Resultado Obtido	Sucesso.

5 Implantação

5.1 Projeto de Implantação

Conforme ilustrado no Diagrama de Implantação e explicado anteriormente, a arquitetura do sistema adota uma abordagem cliente-servidor, onde os componentes de software são mediados pelos protocolos de rede padrão.

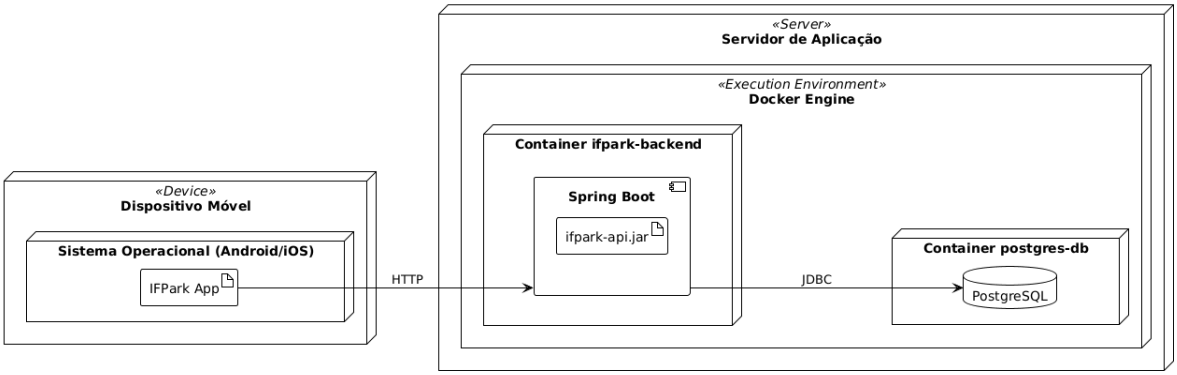

Figura 8: Diagrama de Implantação

O núcleo do processamento está no servidor de aplicação, cuja infraestrutura é gerenciada pelo Docker Engine. A opção pela containerização permite que o backend e o banco de dados operem em ambientes isolados, eliminando inconsistências de dependências nos ambientes de desenvolvimento e produção. Dentro deste host, executam-se dois contêineres principais: o container ifpark-backend, que hospeda a API Spring Boot (empacotada no artefato ifpark-api.jar), e o container postgres-db, responsável pelo banco de dados. A comunicação entre estes serviços ocorre via protocolo TCP/IP e a API do JDBC, garantindo que o banco de dados não precise ser exposto diretamente à rede externa.

Na camada de cliente, o sistema opera em dispositivos móveis com o ambiente de execução do sistema operacional (Android ou iOS). A comunicação entre o dispositivo móvel e o servidor de aplicação é realizada através do protocolo HTTP, permitindo que o sistema funcione tanto via rede local (intranet do campus) quanto via internet.

Para garantir a operacionalidade da solução proposta, foram definidos requisitos mínimos de plataforma. No que tange ao servidor, recomenda-se a utilização de um sistema operacional base Linux (como Ubuntu Server ou Debian) com o Docker Engine (versão 20.10 ou superior) instalado. Em termos de hardware, o servidor demanda um processador dual-core de 2.0 GHz ou superior e, no mínimo, 4 GB de memória RAM para suportar a execução simultânea da JVM e do SGBD PostgreSQL, além de armazenamento em disco suficiente para o crescimento histórico dos dados.

6 Manual do Usuário

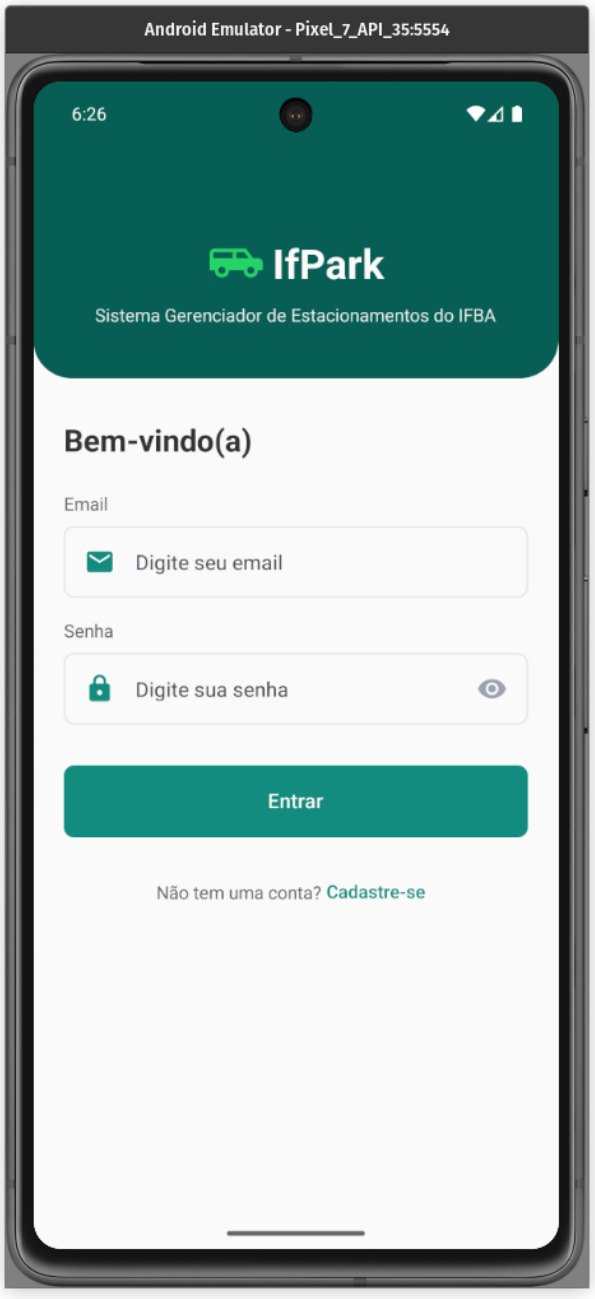

Figura 9: Tela de login do aplicativo

Esta é a tela inicial da aplicação. O usuário pode fazer login caso já tenha uma conta ou realiza seu auto-cadastro, caso seja um usuário do tipo Motorista.

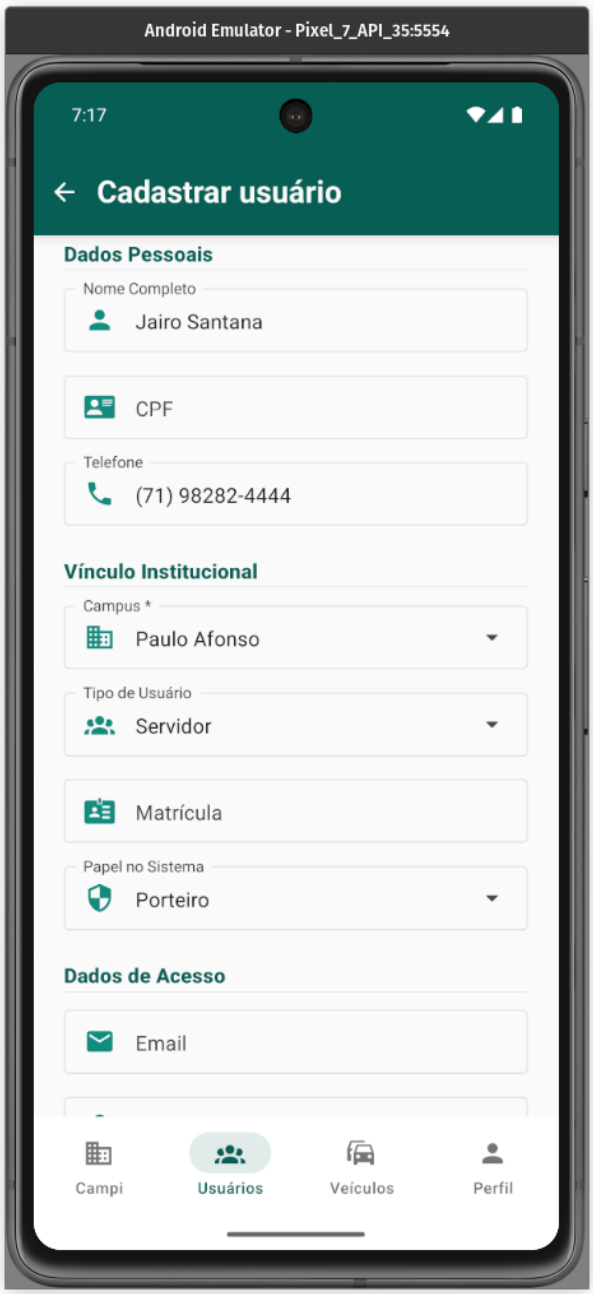

Figura 10: Tela de auto-cadastro

Essa é a tela de auto-cadastro de um motorista. Aqui o usuário deve inserir suas informações pessoais, escolher o campus que faz parte, seu tipo de vínculo, matrícula (se houver), dados de acesso como email e senha, além de endereço. Na tela seguinte, é possível cadastrar seu veículo ou, se preferir, o usuário pode cadastrar seu veículo depois.

Figura 11: Tela inicial do motorista logado

Após realizar o login, o usuário é direcionado para a Tela Principal.Esta é a funcionalidade central para a comunidade acadêmica. Nela, o sistema lista todos os estacionamentos do campus vinculado ao usuário. Para cada estacionamento, é exibido um indicador visual (card) que informa, a quantidade de vagas livres e ocupadas. Se um estacionamento estiver lotado, o indicador ficará vermelho, sinalizando ao motorista que não há disponibilidade naquele setor.

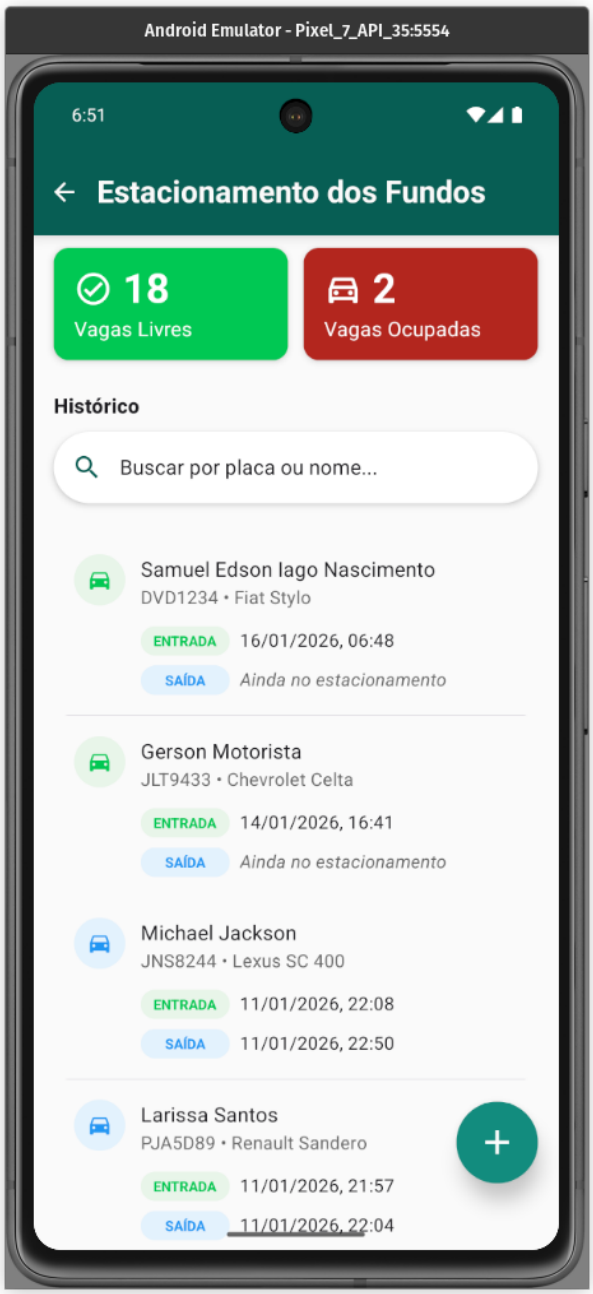

Figura 12: Tela de operação do porteiro

Para os usuários com perfil de Porteiro/Vigia, a interface é focada na agilidade operacional. Nesta tela de Controle de Acesso, o porteiro possui um botão de acesso rápido para registrar ”Entrada”e ”Saída”. O sistema exibe o nome do estacionamento que está sendo monitorado, sua ocupação atual e o histórico das movimentações deste estacionamento. Caso o estacionamento atinja a capacidade máxima, o sistema bloqueará novas entradas para garantir a organização física do local.

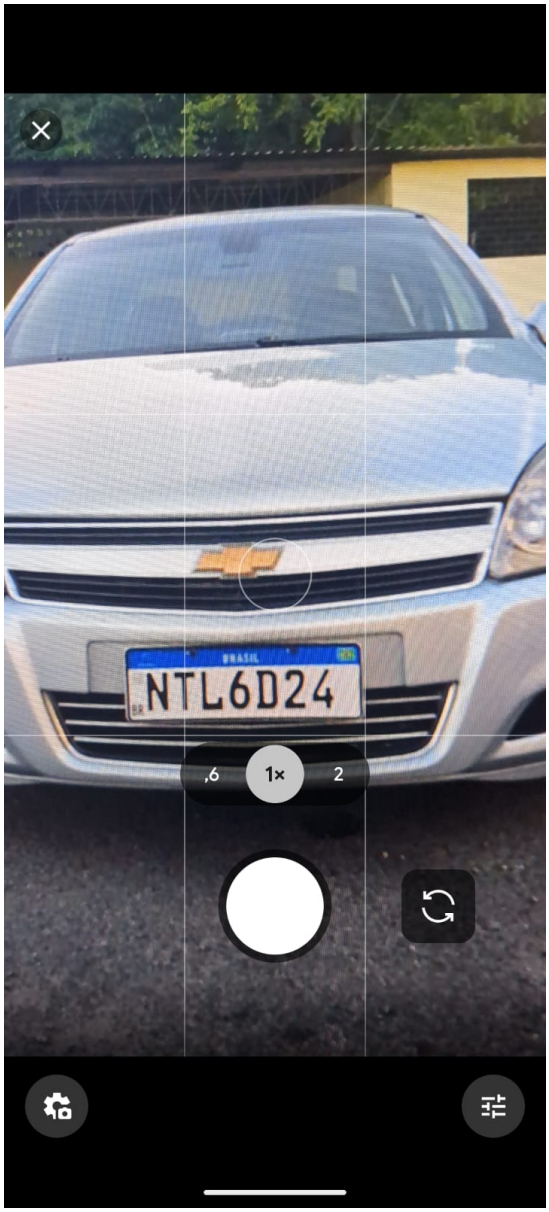

Figura 13: Câmera do Porteiro

Figura 14: Placa Escaneada

Ao selecionar a opção de nova movimentação, o porteiro pode utilizar a funcionalidade de reconhecimento automático de placas. Conforme demonstrado na imagem, basta apontar a câmera do dispositivo móvel para a placa do veículo. O sistema processa a imagem, extrai os caracteres e após confirmar a leitura da placa, preenche o campo de busca. Após a identificação, o sistema valida se o veículo está autorizado e se o motorista está ativo, permitindo a confirmação da entrada com apenas um toque.

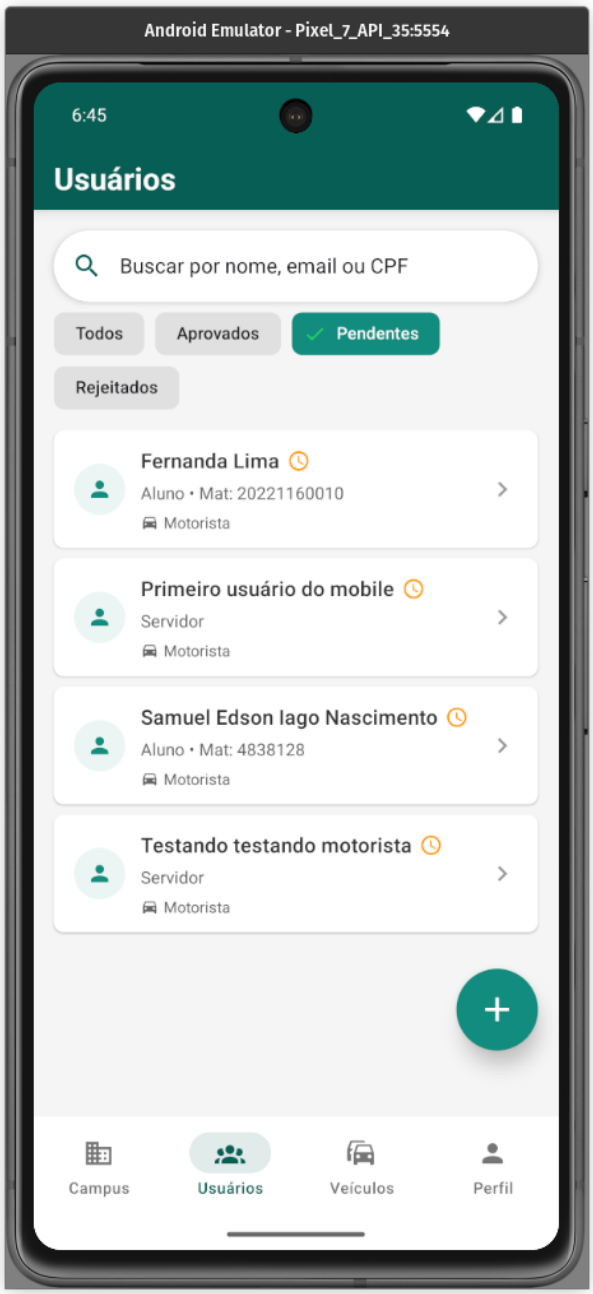

Figura 15: Lista de usuários filtrados por status pendente

A tela de Gestão de Pendências é fundamental para a segurança institucional. Como o auto-cadastro de motoristas gera um status ”Pendente”, cabe ao administrador acessar esta lista, verificar os dados do solicitante e realizar a aprovação ou rejeição do cadastro. Apenas após essa aprovação o motorista terá suas credenciais de acesso liberadas para login.

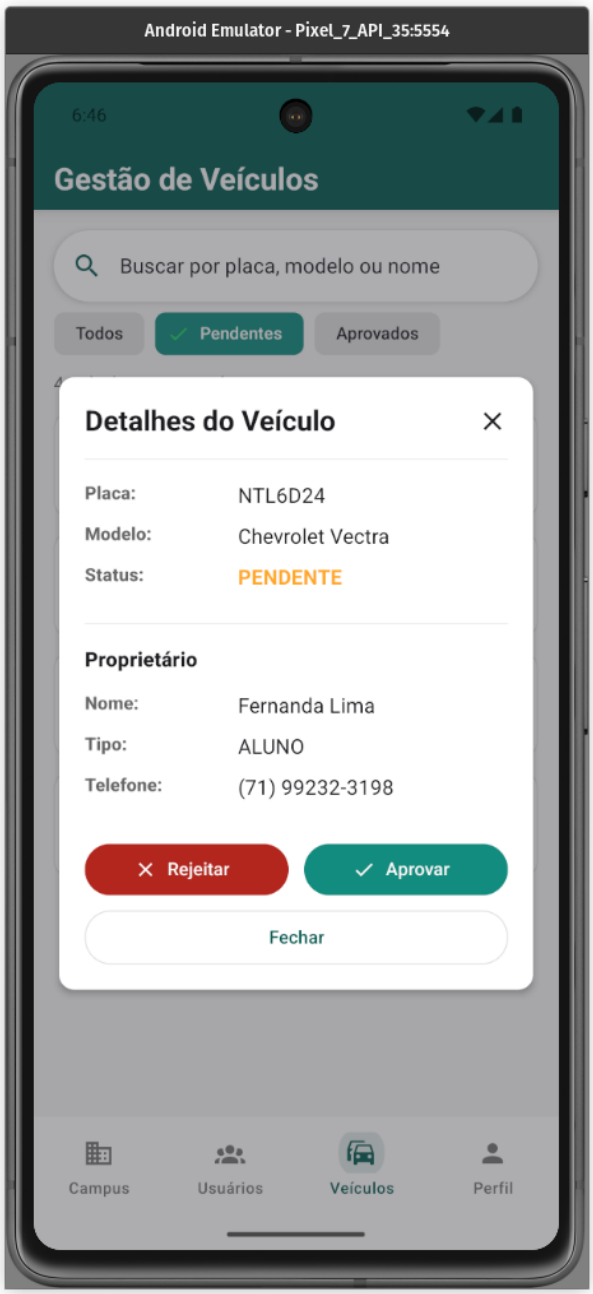

Figura 16: Usuário administrador deve aprovar ou rejeitar veículos

Da mesma forma, o administrador deve validar os veículos cadastrados. Ao clicar em uma solicitação, o sistema exibe os detalhes do veículo (placa, modelo) e do proprietário. O administrador pode conferir as informações e decidir por aprovar o veículo (autorizando sua entrada pelo porteiro no campus) ou rejeitá-lo, podendo inclusive inserir um motivo para a rejeição que será visualizado pelo usuário.

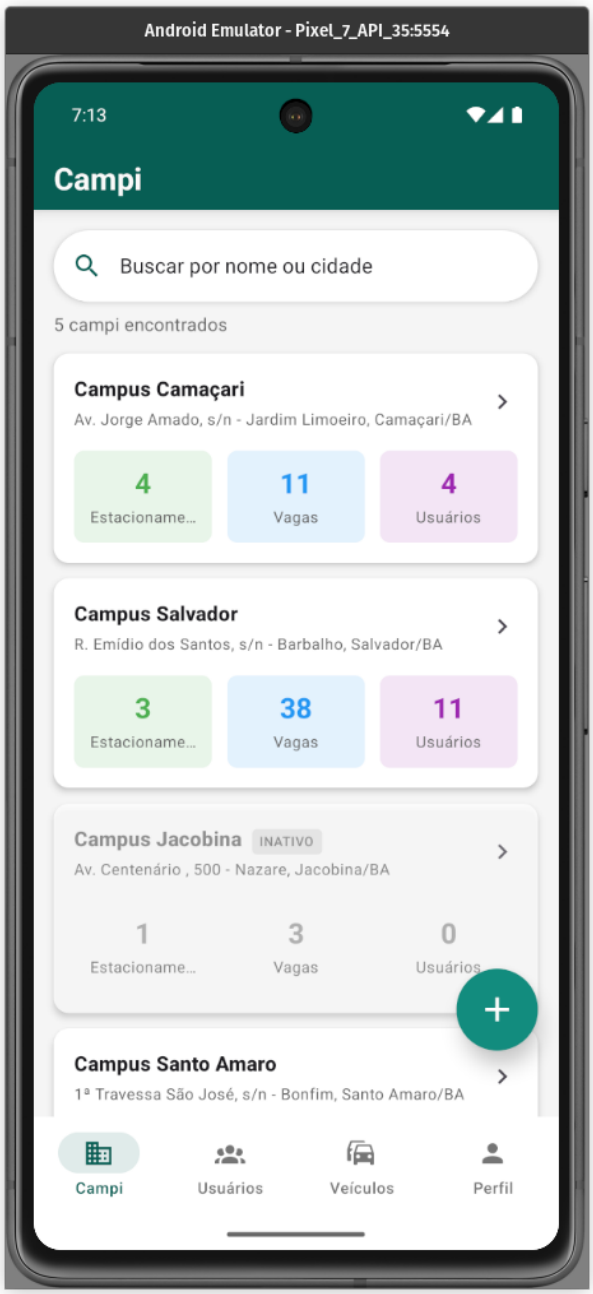

Figura 17: Tela inicial do super-administrador

Por fim, o a super-administrador pode fazer todas essas operações e possui controle total sobre a infraestrutura através das telas de Gestão de Campi. Aqui é possível cadastrar novos campi (para o caso de expansão do sistema para outras unidades do IFBA) e criar ou editar estacionamentos, definindo nomes e, principalmente, a capacidade total de vagas, que servirá de base para os cálculos de lotação do sistema.

Agradecimentos

Ao meu orientador, Manoel Carvalho Marques Neto, agradeço por ter se mostrado disposto a me orientar e por ter validado a ideia deste projeto, além dos feedbacks durante as entregas.

Ao professor Frederico Barboza, pelas ótimas aulas e por ser uma grande inspiração e referência para mim.

À coordenadora do curso, Flávia Maristela Santos Nascimento, expresso minha profunda gratidão. Agradeço não apenas pela gestão acadêmica, mas imensamente pelo apoio que me foi concedido durante um momento difícil. Sua ajuda foi decisiva para que eu não desistisse e pudesse chegar ao final deste ciclo.

Estendo este agradecimento especial ao professor Antônio Carlos dos Santos Souza, que também me ajudou durante esse mesmo período. Obrigado por ter me estendido a mão quando precisei.

Também gostaria de agradecer ao professor Antônio Gabriel Souza Almeida, por ter sido coordenador de um projeto de extensão do qual fiz parte. Tive experiências muito boas enquanto estive nesse projeto e também sou imensamente grato à equipe do Polo de Inovação do IFBA.

Aos meus colegas de turma, obrigado pela troca constante de ideias, pelas conversas e por compartilharem os momentos bons e leves que tornaram essa jornada acadêmica mais prazerosa.

Por fim, agradeço ao Instituto Federal da Bahia e a todos os seus servidores. Minha história com esta instituição começou ainda no ensino técnico integrado e, hoje, após chegar ao ensino superior, olho para trás com orgulho de uma vivência de quase 10 anos. O Instituto foi a base não apenas da minha formação educacional e profissional, mas teve um papel central na formação do ser humano que sou. Sempre terei imensa gratidão por essa instituição.

Referências

1 Redação CORREIO. **Após interdição da Ladeira da Soledade, trânsito está congestionado no Barbalho**. 2011. Jornal Correio. Disponível em: https://www.correio24horas.com.br/transito/apos-interdicao-da-ladeira-da-soledade-transito-esta-congestionado-no-barbalho-0511.

2 COSTA, M. N.; CVITANIć, D. K.; PERKOVIć, L. Ordinal Regression Model of Parking Search Time. **PROMET - Traffic&Transportation**, v. 35, n. 5, p. 689–702, 2023. Menciona ”frustrations and feelings of uncertainty”(frustrações e sentimentos de incerteza) causados pela busca por vagas.

3 MANEPALLI, U.; BHUVANESH, A.; RAM, S. CO2 Emissions Induced by Vehicles Cruising for Empty Parking Spaces in an Open Parking Lot. **Sustainability (MDPI)**, v. 14, n. 7, p. 3742, 2022.

4 ISACA. Audit Trails: Your Best Friends or Your Worst Enemies? **ISACA Journal**, v. 5, sep 2019. Disponível em: https://www.isaca.org/resources/isaca-journal/issues/2019/volume-5/audit-trails.

5 CORPORATION, O. **Java SE 17 Documentação**. 2021. ⟨https://docs.oracle.com/en/java/javase/17/⟩. Acessado em: 10-nov-2025.

6 Pivotal Software, Inc. **Spring Boot 3.0 Reference Documentation**. 2023. ⟨https://docs.spring.io/spring-boot/docs/current/reference/html/⟩. Acessado em: 10-nov-2025.

7 ____. **Spring Security Reference**. 2023. ⟨https://docs.spring.io/spring-security/reference/index.html⟩. Acessado em: 10-nov-2025.

8 ____. **Spring Data JPA Reference Documentation**. 2023. ⟨https://docs.spring.io/spring-data/jpa/reference/html/⟩. Acessado em: 10-nov-2025.

9 The PostgreSQL Global Development Group. **PostgreSQL: The World's Most Advanced Open Source Relational Database**. 2023. ⟨https://www.postgresql.org/⟩. Acessado em: 10-nov-2025.

10 SOFTWARE, R. **Flyway: Database Migrations Made Easy**. 2023. ⟨https://flywaydb.org/documentation/⟩. Acessado em: 10-nov-2025.

11 Springdoc. **springdoc-openapi - Library for spring-boot with openapi 3**. 2023. ⟨https://springdoc.org/⟩. Acessado em: 10-nov-2025.

12 PLATFORMS, I. M. **React Native: Learn once, write anywhere**. 2023. ⟨https://reactnative.dev/⟩. Acessado em: 10-nov-2025.

13 GOOGLE. **ML Kit: Text Recognition (On-device)**. 2023. ⟨https://developers.google.com/ml-kit/vision/text-recognition⟩. Acessado em: 10-nov-2025.

14 React Navigation. **React Navigation: Routing and navigation for Expo and React Native apps**. 2023. ⟨https://reactnavigation.org/⟩. Acessado em: 10-nov-2025.

15 React Native Community. **AsyncStorage: An unencrypted, asynchronous, persistent, key-value storage system for React Native**. 2023. ⟨https://react-native-async-storage.github.io/async-storage/⟩. Acessado em: 10-nov-2025.

16 ARVIDSSON, J. **React Native Keychain: Securely store credentials in React Native**. 2023. ⟨https://github.com/oblador/react-native-keychain⟩. Acessado em: 10-nov-2025.

17 (Beier) Luo, B. **React Hook Form: Performant, flexible and extensible forms with easy-to-use validation**. 2023. ⟨https://react-hook-form.com/⟩. Acessado em: 10-nov-2025.

18 MCDONNELL, C. **Zod: TypeScript-first schema validation with static type inference**. 2023. ⟨https://zod.dev/⟩. Acessado em: 10-nov-2025.

19 The Git Development Community. **Git: fast, scalable, distributed version control system**. 2023. ⟨https://git-scm.com/⟩. Acessado em: 10-nov-2025.

20 JETBRAINS. **IntelliJ IDEA: The Capable & Ergonomic IDE for Java**. 2023. ⟨https://www.jetbrains.com/idea/⟩. Acessado em: 10-nov-2025.

21 CORPORATION, M. **Visual Studio Code: Code Editing. Redefined.** 2023. ⟨https://code.visualstudio.com/⟩. Acessado em: 10-nov-2025.

22 Bruno. **Bruno: Fast and Git-Friendly Open Source API Client**. 2026. Acessado em: 16 jan. 2026. Disponível em: ⟨https://www.usebruno.com/⟩.